(1) This problem aims to cement your understanding of (D)PDAs between the text and the Thursday 4/13 lecture (whose notes have been posted).

(a) The DPDA $M$ accepting $L = \{a^n b^n : n \geq 0\}$ at the bottom of page 3 at the end of the notes has a bug as drawn. The bug (see note added there also) is that if $x = a^n$ where $n \geq 1$, then $x \notin L$, but $M$ still accepts $x$ because the loop at the start state will still see a blank $B$ on tape 2 after reading the $a$’s. Re-design $M$ to fix this bug. The most thematic fix is to take the hint in my lecture to have the first step introduce a bottom-of-stack marker $\wedge$ on tape 2. That first step can stay stationary on the first char $x_1$ of the input $x$ on tape 1. (9 pts.)

Answer: By inserting an extra step of laying down an initial $\wedge$ without consuming the first $a$ on the tape, we distinguish between reading blank+blank at the start state $s$ (where it signifies $x = \epsilon$ which we need to accept) and reading blank+blank after looping on some $a$’s (where it signifies no $b$’s after an $a$ so we need to reject). The $\wedge$ also makes the acceptance condition easier to highlight.

![Diagram](image-url)

The reject conditions, clockwise from lower left, are:

- No $a$’s but some $b$’s;
- Some $a$’s but no $b$’s;
- Too many $b$’s; too few $a$’s; input not in $a^*b^*$. 
(b) Suppose \( c \in \Sigma, A \in \Gamma, \) and a state \( p \) are such that a DPDA \( M \) has an instruction \( (p, (\_A), \ldots) \) but goes into an “infinite loop” upon executing it. That is, \( M \) thenceforth never makes a \( R \) move beyond the char \( c \) on the input tape, and never moves its second tape head left of the cell occupied by the \( A \). (It might push other chars after the \( A \) and the TM version is also allowed to change \( A \) in one step, but it never pops whatever is there.) Then you can replace the instruction by \( (p, (\_A), (\_B), (R)_{B}, d) \) where the state \( d \) acts like a dead state: it cleans up tape 2 by continuing to pop until it’s blank, reads tape 1 until it hits the blank at the end, and finally goes to \( q_{\text{rej}} \) on \( (B)_{B} \). The new machine \( M_0 \) obtained by doing this replacement in all looping cases of \( p, c, A \) gives \( L(M_0) = L(M) \) without ever looping.

Go on to describe how to modify \( M_0 \) into a DPDA \( M' \) such that \( L(M') \) equals the complement of \( L(M) \). For full credit, you should explain that if \( M \) obeys “good housekeeping” on all of its non-looping computations, then \( M' \) obeys it on all computations. “Good housekeeping”—as defined in my lecture notes and applying to all TMs not just PDAs—means that all halting computations end up by erasing tape(s) 2 (and beyond), reading all of tape 1 and ending with the tape head on the cell to the right of the end of the input \( x \) (which could hold a special end-marker $\# \) char instead of \( B \) if you wish—this is an option also in part (a) above), and either rejecting in the state \( q_{\text{rej}} \) or accepting in \( q_{\text{acc}} \). In particular, a “good housekeeping” PDA simultaneously obeys both the “accept by final state” and “accept by empty stack” criteria—well for the latter we technically need to make the final arc(s) to \( q_{\text{rej}} \) refuse to pop the bottom-of-stack marker \( \& \). Finally observe how this shows that the class of DCFLs—unlike the class of CFLs—is closed under complementation. (9 + 3 = 12 pts., for 21 on the problem)

Answer: For every case \( (p, (\_A)/\ldots) \) that causes looping, replace it with the instruction \( (p, (\_A)/(\_B, S), d) \) where \( d \) is a “cleanup” state which finishes reading the input and pops everything on the stack (i.e., on tape 2) prefatory to going to \( q_{\text{rej}} \). This begs the question of how we would detect looping conditions, but since there are only finitely many cases we can declare that whichever ones needed attention got it. The resulting machine \( M' \) is still a DPDA and now enjoys the property that all computations on all inputs end up either at \( q_{\text{acc}} \) or at \( q_{\text{rej}} \). This is the condition needed to ensure that interchanging \( q_{\text{acc}} \) and \( q_{\text{rej}} \) complements the language, giving us a DPDA \( M'' \) such that \( L(M'') = \complement L(M) \). Moreover, because we used the “cleanup” state \( d \) (which now transits to \( q_{\text{acc}} \) in \( M'' \)), all the former looping cases of \( M \) become accepting computations of \( M'' \)—plus if all other rejecting computations by \( M \) observed good housekeeping, they become accepting computations of \( M'' \) that do.

All this establishes that for any DCFL \( L \), we can take some DPDA \( M \) such that \( L(M) = L \), and obtain the DPDA \( M'' \) such that \( L(M'') = \complement L \). This is what it means for the class DCFL os DCFLs to be closed under complementation.

(2) Define \( L = \{wc^mw : w \in \{a, b\}^*, \ m = |w|\} \). This is the same as the language on Assignment 7 (problem 3) except that the final \( w \) is not reversed. Design a two-tape TM \( M \) such that \( L(M) = L \). Note that since \( m = 0 \) is allowed by default, \( \epsilon \) does belong to \( L \), so you will have to handle that case. Also indicate exactly where your \( M \) fails to be a PDA and answer whether you ever needed to keep the input head stationary in any step. (18 + 3 + 3 = 24 pts., for 45 total on the set)

Answer: The following 2-tape TM handles \( \epsilon \) in similar manner to the machine in problem (1) by taking an extra step to lay down an initial \( \& \) marker on tape 2, But it is not a PDA
because of the instructions \((c c R)_{\alpha \alpha L}\) and \((c c R)_{\beta \beta L}\) which kept count of the c’s \textit{without} consuming the information previously stored on tape 2 about the string \(w\) in the left half.

The reject conditions clockwise from start at lower right are:

- Some initial \(a\)’s and/or \(b\)’s in the initial “\(w\)” part but no \(c\)’s follow;
- Too few \(c\)’s; too many \(c\)’s (on hitting the \(\land\) prematurely on tape 2);
- Mismatch of \(a\) vs. \(b\); right-hand side string too long; right-hand side string too short.

The input head stayed stationary just twice: once at the beginning and once when taking one step needed to go from “push mode” to “check mode” upon hitting the first \(c\). Otherwise, it marched forward at every step—so not only did it finish in \(O(n)\) time, it almost operated “in real time” the way a DFA does.

Note finally that the 2-tape TM corresponds closely to the following natural prose description of an algorithm for deciding the language:

“If the tape is empty, accept. Else copy an initial substring \(w\) of \(a\)’s and \(b\)’s to tape 2 until the first \(c\)—if there is no \(c\) then reject. Count the number of consecutive \(c\)’s from that point on. If it doesn’t exactly match the length of the substring \(w\), reject. Else, rewind on tape 2 to the beginning of \(w\) (this will already have happened when counting \(c\)’s against \(|w|\)) and start comparing \(w\) with whatever comes after the last \(c\). Accept if and only if whatever follows exactly equals \(w\).”

Starting with Chapter 4, it will be OK to abbreviate Turing machines with such prose.